8. ソ　ー　ト　法†

溝　口　徹　夫††

1. ま え が き

他のいくつかの問題解決法がそうであるように，ソート法は計算機の出現以前から存在したものである．ここでことさらソート法をとり上げるのは，次の2つの理由による．第1にはソート法，特に外部記憶を利用するもの，は実用的価値が高く，新装置開発には対応した課題の解決法の必要があること．第2にはソートはその目的が単純明解であること（与えられたデータを一定の順序に並び換える），アルゴリズムの考案，解析を行いやすい内容を持つこと（データ利用に不確定要素を含まない等）による．実用性と理論的解析の容易さの2面からすでに多くのアルゴリズムに関する提案，解析が報告されてきたし，現在も続いていると言えよう．ソート法の新しい環境（たとえばVLSI 利用）での検討も行われている．アルゴリズムを云々する時にはまずどのような具体的手法を用いるかそしてその手法の利点は何なのかを明らかにする必要がある．本文でソート手法として，主記憶のみを使う内部ソート法，外部記憶をも使う外部ソート法等について述べ，次に各種ソート手法の評価について述べることとする．分類は文献 1) に準じた．

2. 内部ソート法

ソーティングを実施する条件で，すべてのデータが主記憶に入り切る（すべてのデータが1回のアクセスで単独に，同一コストで得られる）場合をここで扱う．たとえば，トランプのカード（ハートが13枚），順不同で，(5, 3, 1, 2, 8, 9, Q, 4, 6, 10, K, J, 7) と重ねられているが，これを数の小さい順に並べたいとしよう．与えられた入力箱の内容を物理的に順序を変えて出力箱での（大小）順序に変換するのがソート処理である．(**図-1** 参照) 入力箱と出力箱は別々でもよいし，同じ箱を使ってもよく，またソートは部分的な順序がえを繰り返し行うこともある．

方法 1：挿入法（**図-2** 参照）

入力箱から入力データの選び方：入力データは一定の位置（たとえば未処理データの物理的先頭位置）から選ぶ．

ソート手法：選んだデータがその時点までの出力箱の中で位置すべき順番を見つけ，その位置に挿入する．

方法 2：選択法（**図-3** 参照）

入力データの選び方：入力データの中で未処理分のうち，最小のものを見つけて来る．

ソート手法：リストの最後にデータを追加する．

方法1と方法2は似かよっているが，相違点は負担のかけ方を出力時（方法1）にかけるか，入力時（方法2）にかけるかである．

方法1と方法2での挿入や選択で，出力箱の大部分

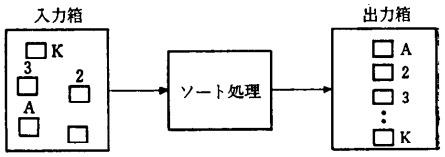

図-1　内部ソート法

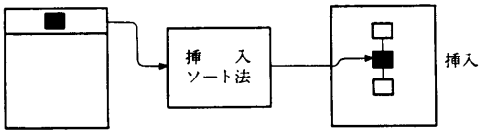

図-2　方法 1；挿入法

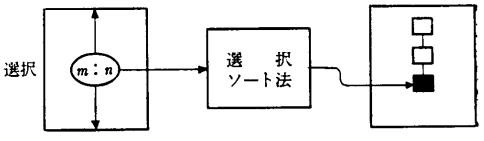

図-3　方法 2；選択法

† Sorting Methods by Tetsuo MIZOGUCHI (Mitsubishi Electric Crop. Information Systems and Electronics Development Lab.).

†† 三菱電機(株)情報電子研究所

のデータの移動や入力箱での全データ比較による最小値選択を入力データごとに繰り返すのは非能率である．そこで挿入選択をサブ問題として効率化する必要がある．(詳細は野下 2)，溝口 3) 参照)．

入力箱から最小値を選ぶ方法をとる選択ソート法では，先頭の入力データの比較の歴史は以後のデータの比較に活かされないので，毎回無駄な比較をすることになる．そこで入力箱のデータの特徴を一括して構造化し，最小値を選びやすくするものとして，**heap**(ヒープ) を用いる方法を示そう (Horowitz-Sahni[4])．heap を二分木 (binary tree) 構造で表現する (**図-4** 参照)．説明の都合上以下では二分木を用いる．heap のある node (節) の双方の子 node は親 node よりもデータ値は大きいものを持つ．前記の例で 13 個の順不同のデータの初期格納位置 1〜13と heap の構造は図-4 に示す対応になる (例位置 4 の子 node は 8 と 9 に入っている)．**図-5** の初期状態で下のレベルから順に heap の条件を満すように親子の node の値を交換する．heap の subtree の頭は必ず最小値が来ているので出来上った heap の頭には必ず最小値が来る．途中で親子の交換を行った後，子と孫の交換が逐次下方に発生し得るが，それも高々 heap の木の高さで終ることは明らかである．図-5 において 1 個データを抜きとれば，heap の修復が必要である．

方法 3：交換法 (**図-6** 参照)

入力データの選び方：一定の位置から順次入力を繰り返し行う．

ソート手法：比較を行い，順序が逆であれば，位置を交換する．

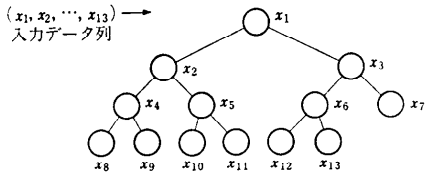

図-4 入力データ列と二分木の対応

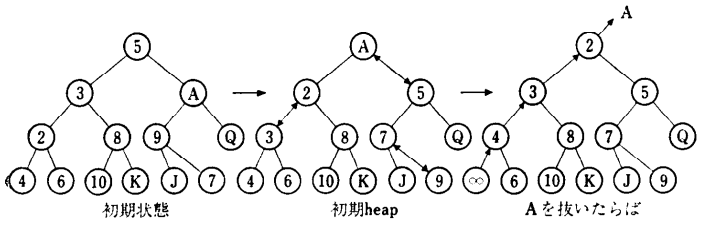

図-5 heap を用いた最小値選択

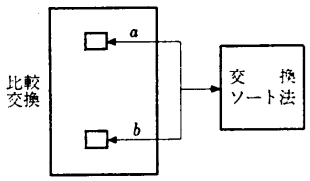

図-6 方法 3；交換法

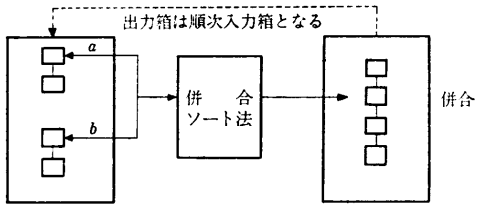

図-7 方法 4；併合法

方法 4：併合法 (**図-7** 参照)

入力データの選び方：隣接する入力データ列を複数列とって来る (図では入力データ列は 2 個)．

ソート手法：各々はすでに順序づけられているデータ列複数個内のデータを比較 (互いに先頭から比較し合えばよい) を行い，1 つの列に併合 (merge) する．

この 2 つの方法の共通点は入力データを一通り走査し，データの並びを変えるが，一回の走査ではソートは終了しないので処理は繰り返えされる．方法 3 は比較，交換の対象を広域的に検索し，順次領域をせばめて領域内を順序化するが，方法 4 では比較の対象を近傍に限定し，局所的かつ部分的な順序づけを順次広域化するので，各々 top-down, bottom-up の手法といえる．方法 3 の 1 つである **Quick sort** の例を**図-8** に示す．図で丸印は交換の起きたデータを示す．与えられたデータ列の左端を固定し，右端のデータから順に，左端のデータよりも小さいものがあるか否か捜し，最初に見つかったものと左端のデータを交換．次に交換を行った区間で，右端を固定し，左端から順に前記の手順を繰り返す．左右の検索/交換が出合うまで繰り返す．図の例では左右が出合った点より左では 5 以下，右では 5 以上のデータが並んでいる．この 2 領域について以後交換法を繰り返し適用する．同一データを併合法を用いると**図-9** のようになる．

方法 5：分配法

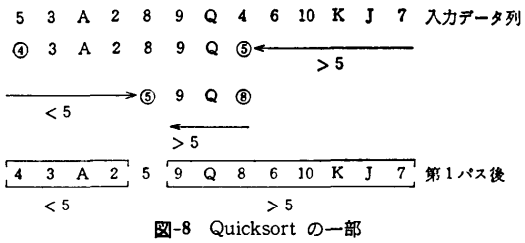

図-8　Quicksort の一部

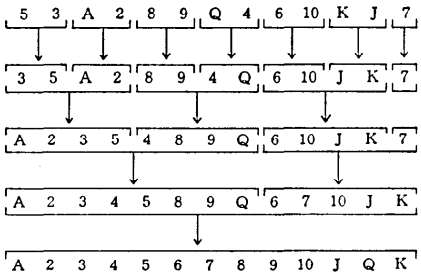

図-9　併　合　法

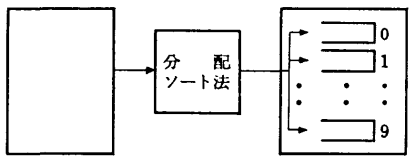

図-10　方法 5；分配法

かつて多用されたカードソータはこのソート法による．

入力データの選び方：入力データを先頭から入力する．

ソート手法：下位の桁のデータ値に相当する箱（十進データでは 0～9 の値に対応する箱）にデータを分配する．分配したものをまとめ，実に上位桁について繰り返し分配を行う．

3. 外部ソート法

本節では主記憶に加えて外部記憶を必要とする場合のソート手法について述べる．各種事務処理において，外部記憶を使ったファイルの活用が行われている．ファイル利用に伴って外部ソートも多用される．外部ソートの構成を図-11 に示す．外部ソートは内部ソートを含んだ形になっている．そこで外部ソートの問題は何かというと，外部記憶の特徴（特に物理的特徴）を活すためのソート方策を講じることである．外部記憶装置として従来，磁気テープと磁気ディスクが用いられてきた．外部記憶の特徴には次のものがある．

（1）　外部記憶にあるデータを主記憶に移して，ソート処理に提供するまでの所要時間は平均 30ミリ秒位であり，内部ソートでのデータの主記憶常駐の場合と比べて大きな差がある．

（2）　外部記憶と主記憶間のデータはブロックで一括転送するので，一括転送された後のデータに対しては（1）の条件は成りたたない．

（3）　外部記憶装置は一般に順次（sequential）なデータの読み書きを得意とする．（磁気ディスクはランダムアクセスをもある程度効率よく実施する）．また磁気テープのようにソートに同時に必要な台数に制限がつく場合もある．

以上の特徴から言えることは，内部ソートの場合での比較回数という評価指標に比べて，外部ソートでは，

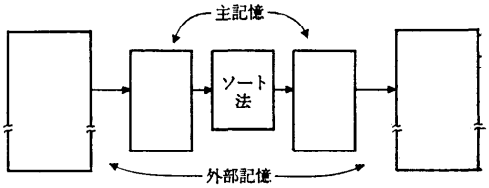

図-11　外部ソート

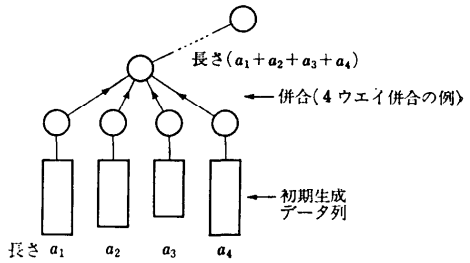

図-12　マルチウエイ併合法

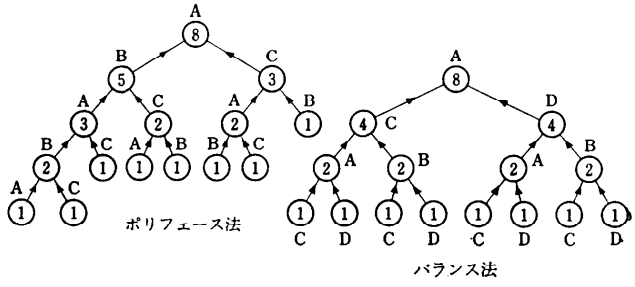

図-13　ポリフェーズ法とバランス法

入出力実行回数が大きな役目を果す.

外部ソート手法の特徴を列挙すると（図-12 参照）

（a） ブロックで一括入出力するためには，まず最初に出力するデータは内部ソート法を使って，ソートされている部分列の長さがなるべく長いものを作る必要がある.

（b） 外部記憶の物理的特性上から初期データ列生成以後は併合法を用いることが多い．次の入出力が現在の入出力を行った外部記憶上での物理的位置の隣接位置に対して行われることがこの方法で可能となるためである.

（c） 併合はそのウエイ数（同時に併合される入力列の数）が多い程，パス数は減るので有利である．しかし，各入力データ列から最低1ブロックずつ主記憶内にデータを保持しておく必要があるので，主記憶量からウエイ数の上限が与えられる．入力をスムーズに行うためのダブルバッファ法をとると更に所要記憶量が増す.

磁気テープを用いた併合法では，図-13 に示すごとく一般には最低4台のテープを同時に用意しておかねばならないが，これを3台ですませる方法でフィボナッチ数を用いたポリフェーズ法が使われる．図で丸の中の数字はブロック数，ABCD は各々4台までのテープ装置の名を示す．図では同じ8ブロックのデータのポリフェーズ法とバランス法を各々3台，4台のテープを用いて併合する場合の併合の流れを示す．ではテープが1台または2台の時にはどうすればよいのであろうか？ 磁気ディスクでは少し環境は異なる．（例えばオーバラップシークのためのワークファイル領域の割当等）2台の場合を Smith-Floyd[5] 溝口[6] が，1台の場合を Floyd[7]，Karp[8] らが扱っている．テープソートのディスクソートへの移行がすすんでいるが，外部ソートを論理的に解析するためにテープソートは役立つ．詳細は省くが n-tape Turing machine の complexty の議論と n-tape ソートの対比は興味深い．視点を変えて，外部記憶を仮想的に見た時のソート法や，キーのみをとり出してソートを行う手法等も考案されている.

4. 並列ソート法

並列ソートを研究対象とする動機は既に述べたように，第1には並列処理一般でなく，特定の利用での検討対象とすることによる並列処理そのものの理解の助けとなる面と，第2に実用的と予想される面への適用，たとえばVLSI化，データベースマシンの要素としてのソート機能素子の実現，を目指す面とがある．特に後者は商用化まではいたっていないが，種々の試みがなされている[9),10)].

5. アルゴリズム解析

ソート法は有限個のデータ n 個を与えられ，それの並び換えを行うもので，必ず有限時間で終了し得るものである．そこでソートアルゴリズムの解析では，正しくソートを行うか否か，その処理をいかに早く，少ない所要記憶量で終了するかの評価を行うことが大切となる．アルゴリズム解析をソート時間の下限，上限および時間-記憶容量積の限界に分けて述べる.

5.1 ソート時間の下限

あるモデルを設定した時，そのモデルで短縮できる時間の下限についてここで述べてみよう．まず内部ソートについては，「2個のデータの大小比較」を基本動作とし，比較の結果は大または小のいずれかとする．必要に応じてデータの交換等は行うとする．今データ n 個が与えられ，その内容（順序づけ等）について何ら予備知識を持たない時，どのようなデータの与えられ方をしようとも，比較を行ってソートを終了する際比較回数をなるべく少なくしようと試みた時，一体どの位まで少なくできるのであろうか？ 一体その限界はいくつなのであろうか？ ここで大切なことは入力データについて最悪の事態を想定している所であり，また処理側には最良を要求しているし，特定の処理法の比較回数ではなく，仮想的最良アルゴリズムでの比較回数を問題としている所である.

n 個の入力データの並び方は $n!$ 通りあるが，ソート法とは，その $n!$ の可能性の中から比較という動作を経て，正しい順序づけを持つ1個に対象を絞ることと考えてよい．比較動作は対象となる可能性の数を大と小に等分することが望ましい．なぜなら等分でない時には多く分けられた方の可能性についてはより多くの比較を必要とし，与えられた入力データ列によっては最小比較数にはならないからである．比較回数の下限は $n!$ 個の葉を持つ完全二分木の高さ；

$$\lceil \log_2 n! \rceil = n\log_2 n - n/(\ln 2) + \frac{1}{2}\log_2 n + Q(1)$$

で与えられる.

外部ソートにおいてはアルゴリズムの効率は入出力回数として勘定するのが妥当と言える．モデルとしては K 階あるビルのエレベータが各階に b 人，全体で

は Kb 人の人間を希望行先通りに届けるために停止する回数を入出力回数に対応させる．次に行先を同じくする人間が一緒になっている度合を togetherness rating という情報理論的量で表現し，1回の停止で増分可能な最大量から，必要停止回数を求めるものである[1),7)]．その解析から得られた結果は次の通りである．エレベータに入りきる人間の数を m とする

$(K.b.\log_2 b)/m$

5.2 各種ソートアルゴリズムの時間上限

時間上限を解析する場合の前提として，解析は入力データが提供したアルゴリズムに最悪の形で働く場合か，それとも平均的入力（または入力データの分布）を想定にする場合かで当然結果が異なる．前者の最悪ケースに対して後者の平均ケースは一般に解析は難しい．例に示した heapsort は最悪ケースでは heap 生成には $O(n\log_2 n)$ の比較が必要だが，平均ケースで必ずそれだけの比較が必要かどうかはより複雑な解析を必要とする．Quick sort は $O(n^2)$ の比較を必要とする例を容易に作り得るが，平均的にはむしろ高速なものとされている．merge sort では最悪，平均とも比較の回数は同じで $O(n\log_2 n)$ である．

外部ソートにおいては，ブロック数を n とする時，外部記憶台数が1台の時は $O(n^2)$，2台以上であれば $O(n\log_2 n)$ の入出力回数で実行終了できるアルゴリズムが存在する[7),8),5),6)]．

5.3 比較的最近の成果

筆者の知る範囲での最近の成果をいくつか拾ってみる．

(a) $X+Y$ (集合 X の要素と Y の要素の和の集合) のソートに $O(n^2\log_2 n)$ が必要[11)]．

(b) ソートに必要な時間・容量の積は $\Omega(n^2/\log_2 n)$ である[12)]．

6. あ と が き

本文では非常に大まかなソートアルゴリズムの話に終始したが，個々のアルゴリズムの詳細な議論，およびオーダレベル以下での解析を本来は行わねばならない．今後の課題としては，より複雑な条件下でのソート法，ハードウェアの特徴をした並列ソート法等に検討の余地が残っていそうである．

参 考 文 献

1) Knuth, D. E.: "The art of computer programming Vol. 3 Sorting and searching" Addison-Wesley (1973).
2) 野下浩平："セレクション法" 本号.
3) 溝口徹夫："“B-tree” によるデータ管理" 情報処理, 21, 7 (昭55年7月).
4) Horowitz, E. and S. Sahni: "Fundamentals of data structures" Computer Science Press, (1976).
5) Smith, A. J. and Floyd, R. W.: "A linear time two tape merge" IPL Vol. 2, No. 5 (Dec. 1973).
6) 溝口徹夫："磁気ディスク装置2台による予測マージ法" 信学会論文誌 D, 57-D, 10 (昭和49年10月).
7) Floyd, R. W.: "Permuting information in idealized two-level storage" in *Complexity of computer computations*, Plenum (1972).
8) Karp, R. M.: "Two combinatorial problems associated with external sorting" in *Combinatorial algorithms*, R. Rustin (ed.) Algorithmics (1973).
9) Thompson, C. D. and H. T. Kung: "Sorting on a mesh-connected parallel computer" Proc. of 8th ACM STOC (1976).
10) Tanaka, Y. et al.: "Pipeline searching and sorting modules as components of a data flow database computer" IFIP 80 (1980).
11) Harper, L. H. et al.: "Sorting X+Y", CACM, 18, 6 (1975).
12) Borodin, A. and Cook, S.: "A time-space tradeoff for sorting on a general sequential model of computation" Proc. of 12th ACM STOC (1980).

(昭和57年11月29日受付)